The Benefits of Incremental Reasoning in OWL EL

Yevgeny Kazakov and Pavel Klinov

The University of Ulm, Germany
{yevgeny.kazakov, pavel.klinov}@uni-ulm.de

Abstract. This demo will present the advantages of the new, bookkeeping-free method for incremental reasoning in OWL EL on incremental classification of large ontologies.[1] In particular, we will show how the typical experience of a user editing a large ontology can be improved if the reasoner (or ontology IDE) provides the capability of instantaneously re-classifying the ontology in the background mode when a change is made. In addition, we intend to demonstrate how incremental reasoning helps in other tasks such as answering DL queries and computing explanations of entailments. We will use our OWL EL reasoner ELK and its Protege plug-in as the main tools to highlight these benefits.

1 Introduction

The $\mathcal{EL}$ family of Description Logics (DLs) are tractable extensions of the DL $\mathcal{EL}$ featuring conjunction and existential restriction. It is the formal basis of the OWL EL profile [2] of the Web ontology language OWL 2 specifically aimed at applications that require management of large terminologies, which is common in biology, health care and life sciences. Ontology classification is the core reasoning task used by such applications. It requires computing all entailed (implicit) subsumption relations between atomic classes. Specialized $\mathcal{EL}$ reasoners, such as CEL [3], ELK [4], jcel [5], and Snorocket [6] are able to compute the classification for ontologies as large as SNOMED CT [7] with about 300,000 axioms. Classification plays the key role during ontology development, e.g., for detecting modeling errors that result in mismatches between terms. But even with fast classification procedures, frequent re-classification of ontologies can introduce significant delays in the development workflow, especially as ontologies grow over time. This motivates development of *incremental reasoning* methods which do not recompute the entire class hierarchy after local changes but manage to incorporate the changes incrementally.

The demo will present a novel incremental reasoning procedure implemented in ELK 0.4.0 and its positive impact on re-classification and related reasoning problems.[2]

1.1 State of the Art

Several incremental reasoning procedures have been developed for ontology languages. Most procedures maintain extra information to trace conclusions back to the axioms in order to deal with axiom deletions.

[1] This submission complements the accepted research paper which explains the developed incremental reasoning method in full technical detail [1].

[2] A screencast will be available at `https://code.google.com/p/elk-reasoner/`.

The Pellet reasoner [8] implements a technique called *tableau tracing* to keep track of the axioms used in tableau inferences [9]. Tracing maps tableau elements (nodes, labels, and relations) to responsible axioms. Upon deletion of axioms, the corresponding elements get deleted. This method is memory-intensive for large tableaux and currently supports only ABox changes.

The *module-based* incremental reasoning method does not perform full tracing of inferences, but instead maintains a collection of modules for derived conclusions [10]. The modules are (not necessarily minimal) subsets of the ontology that entail the respective conclusion. If no axiom in the module was deleted then the entailment is still valid. Unlike tracing, the method does not require changes to the reasoning algorithm, but still incurs the cost of computing and storing the collection of modules.

Managing the extra information such as traces or modules, broadly referred to as *bookkeeping*, typically incurs only a linear overhead. However, even that can substantially hurt user experience on large ontologies such as SNOMED CT.

1.2 Common Use Cases for Incremental Reasoning

The most frequently occurring scenarios when incremental reasoning is beneficial can be summarized as follows:

Continuous Classification The typical ontology development workflow consists of adding, removing, or modifying axioms and occasionally invoking a reasoner to classify the ontology. The latter is done to verify that the changes do not trigger any unwanted entailments and that all desirable entailments are there. Since classification tends to get slower as the ontology grows large, the ontology engineers often do it "offline" after a considerable set of changes has been accumulated. This is sub-optimal because if an error did occur it can become a needle in the haystack to find. A better approach is to classify the ontology continuously in the background mode, i.e., similarly to how modern IDEs continuous compile software's source code and immediately point out errors. Of course, this approach requires a fast, *incremental* incorporation of changes.

DL Queries Certain applications make use of a form of queries, also called *DL Queries*, based on complex class expressions. Every DL query is a class expression for which inferred superclasses, subclasses, or individuals need to be computed. One example of such application is the Virtual Fly Brain project.[3] DL Queries can be straightforwardly implemented by introducing fresh class names. Suppose superclasses need to be computed for a complex class C. Then one can introduce a fresh class name C', add the axiom $C' \sqsubseteq C$ to the ontology, and then re-classify it so that C' finds its place in the class hierarchy. Obviously, the last step is better be implemented incrementally.

2 Incremental Reasoning in ELK

This section briefly describes the main aspects of the incremental reasoning procedure implemented in ELK. Full technical details, including the $\mathcal{EL}^+$ inference rules, algo-

[3] `http://www.virtualflybrain.org/`

rithms, proofs, and experiments can be found in the research track paper [1]. Also, the interested reader can run the code examples provided on the ELK Web page.[4]

There are two main ideas behind our method. The first has been borrowed from the known DRed (over-delete, re-derive) method for maintaining materialized views in databases [11]. When an axiom is deleted or modified, conclusions of all $\mathcal{EL}^+$ inferences in which the axiom was used (as a side-condition) are deleted. Then the same happens to conclusions of all inferences which use deleted conclusions as premises (until a fixpoint). It is well-known that it may lead to over-deletion since some conclusions may have alternative derivations. Our second idea is based on *partitioning* of all conclusions to identify those which may need to be restored. Crucially, partitions are *not* stored, as modules or traces, during the forward classification and do not incur any overhead. Due to space limitations, we only illustrate the method on a small example.

Example 1. Consider the following $\mathcal{EL}^+$ ontology $\mathcal{O}$:

(ax1): $A \sqsubseteq \exists R.B$ (ax2): $\exists R.B \sqsubseteq C$ (ax3): $B \sqsubseteq \exists S.A$

(ax4): $\exists S.C \sqsubseteq C$ (ax5): $C \sqsubseteq D$ (ax6): $\exists S.\top \sqsubseteq D$

One can see that $\mathcal{O}$ entails the following atomic subsumptions: $A \sqsubseteq C$, $B \sqsubseteq C$, and $B \sqsubseteq D$ (we omit the intermediate inferences). Now, let us see what will happen if (ax4) is deleted. The axiom was used to derive $B \sqsubseteq C$ (together with (ax3) and another conclusion $A \sqsubseteq C$) which is retracted first. Then $B \sqsubseteq D$ is also deleted since it was produced by an inference which had $B \sqsubseteq C$ as a premise (using (ax5)). After that the conclusions whose left hand-side is one of $\{\exists S.C, \exists S.A, B\}$ are repaired (intuitively, these are classes whose superclasses changed during the deletion). The repair stage re-applies the inference rules w.r.t. the remaining axioms and restores the conclusion $B \sqsubseteq D$ using (ax3) and (ax6). Other conclusions, e.g., for A or C on the left, are intact.

Our experiments demonstrate that in practice the partitioning tends to be pretty fine and the changes are rather local, i.e., not many partitions need to be repaired. This is the reason why for large ontologies, such as SNOMED CT, incremental classification is 10–40 times faster than full classification, making re-classification nearly instantaneous (see [1] for more details and a comparison with the modularity-based method).

3 Structure of the Demonstration

Finally we describe what we intend to demonstrate during the demo session. Our general goal is to demonstrate performance gains resulted from incremental reasoning to give ontology developers a sense of how their user experience can be improved.

3.1 Continuous Classification

We will use large ontologies, such SNOMED CT, an $\mathcal{EL}^+$ version of GALEN, or others suggested by participants, to demonstrate the sub-second re-classification for a typical ontology editing workflow. We will use the ELK Protege 4+ plug-in[5] which allows users

[4] `https://code.google.com/p/elk-reasoner/wiki/IncrementalReasoning`

[5] Available at `https://code.google.com/p/elk-reasoner/downloads/list`

to turn incremental reasoning on and off to highlight the performance differences. We plan to prepare some changesets, including those introducing errors, e.g., class unsatisfiability, but will also let the participants make their own changes to ontology axioms.

3.2 Fast DL Query Answering

We will demonstrate fast answering of DL queries based on incremental reasoning. For large ontologies, such as SNOMED CT, it will be visible that answering a single query takes considerably less time than re-classification. We plan to use the DL Query plugin for Protege for interactive query answering (so that participants can enter their own queries). To people more interested in answering DL queries via a programming or Web interface, we will show how a simple Web service can handle parallel DL queries posted over HTTP (by computing the corresponding $\mathcal{EL}$ saturations incrementally).

References

1. Kazakov, Y., Klinov, P.: Incremental reasoning in OWL EL without bookkeeping. In: International Semantic Web Conference. (2013) to appear.
2. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C., eds.: OWL 2 Web Ontology Language: Profiles. W3C Recommendation (27 October 2009) Available at `http://www.w3.org/TR/owl2-profiles/`.
3. Baader, F., Lutz, C., Suntisrivaraporn, B.: Efficient reasoning in $\mathcal{EL}^+$. In Parsia, B., Sattler, U., Toman, D., eds.: Proc. 19th Int. Workshop on Description Logics (DL'06). Volume 189 of CEUR Workshop Proceedings., CEUR-WS.org (2006)
4. Kazakov, Y., Krötzsch, M., Simančík, F.: Concurrent classification of $\mathcal{EL}$ ontologies. In Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E., eds.: Proc. 10th Int. Semantic Web Conf. (ISWC'11). Volume 7032 of LNCS., Springer (2011) 305–320
5. Mendez, J., Ecke, A., Turhan, A.Y.: Implementing completion-based inferences for the $\mathcal{EL}$-family. In Rosati, R., Rudolph, S., Zakharyaschev, M., eds.: Proc. 24th Int. Workshop on Description Logics (DL'11). Volume 745 of CEUR Workshop Proceedings., CEUR-WS.org (2011) 334–344
6. Lawley, M.J., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In Taylor, K., Meyer, T., Orgun, M., eds.: Proc. 6th Australasian Ontology Workshop (IAOA'10). Volume 122 of Conferences in Research and Practice in Information Technology., Australian Computer Society Inc. (2010) 45–49
7. Schulz, S., Cornet, R., Spackman, K.A.: Consolidating SNOMED CT's ontological commitment. Applied Ontology **6**(1) (2011) 1–11
8. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. of Web Semantics **5**(2) (2007) 51–53
9. Halaschek-Wiener, C., Parsia, B., Sirin, E.: Description logic reasoning with syntactic updates. In: OTM Conferences (1). (2006) 722–737
10. Cuenca Grau, B., Halaschek-Wiener, C., Kazakov, Y., Suntisrivaraporn, B.: Incremental classification of description logics ontologies. J. of Autom. Reason. **44**(4) (2010) 337–369
11. Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining views incrementally. In Buneman, P., Jajodia, S., eds.: Proc. 1993 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C., ACM Press (May 26-28 1993) 157–166